A Finite Difference Domain Decomposition Method Using Local Corrections for the Solution of Poisson's Equation

Gregory T. Balls
University of California at Berkeley
Phillip Colella
Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory

We present a domain decomposition method for computing finite difference solutions to the Poisson equation with infinite domain boundary conditions. Our method is a finite difference analogue of Anderson's Method of Local Corrections. The solution is computed in three steps. First, fine grid solutions are computed in parallel using infinite domain boundary conditions on each subdomain. Second, information is transferred globally through a coarse grid representation of the charge, and a global coarse grid solution is found. Third, a fine grid solution is computed on each subdomain using boundary conditions set with the global coarse solution, corrected locally with fine grid information from nearby subdomains.

There are three important features of our algorithm. First, our method requires only a single iteration between the local fine grid solutions and the global coarse representation. Second, the error introduced by the domain decomposition is small relative to the solution error obtained in a single-grid calculation. Third, the computed solution is second-order accurate and only weakly dependent on the coarse grid spacing and the number of subdomains. As a result of these features, we are able to compute accurate solutions in parallel with a much smaller ratio of communication to computation than more traditional domain decomposition methods.

We present results to verify the overall accuracy, confirm the small communication costs, and demonstrate the parallel scalability of the method.

Key Words: finite difference methods; multigrid methods; domain decomposition; local corrections; potential theory

1. INTRODUCTION

Solutions to Poisson's equation have strong locality properties: the field induced by a localized charge distribution is an analytic function away from the charge dis-

tribution. We expect that, for a charge distribution with multiple scales, nonlocal coupling should be representable by a much smaller number of computational degrees of freedom than the number of degrees of freedom required to represent the local influences of the solution.

This observation leads to domain decomposition as a natural strategy for computing solutions to Poisson's equation on parallel computers. One decomposes the computational domain into disjoint subdomains, computes local solutions corresponding to the charge distributions on each subdomain, and computes the far-field coupling using a reduced description of the data. In the case of particle representations of solutions to the Poisson equation such as the Fast Multipole Method (FMM) [12] and the Method of Local Corrections (MLC) [1], [2], this approach has been used to dramatically reduce the cost of computing the potential induced by a collection of charged particles. These algorithms map easily to parallel architectures using domain decomposition [6], [4], leading to efficient parallel algorithms with low communications requirements.

In the case of grid-based discretizations of the Poisson equation, parallel domain decomposition methods have typically led to iterative methods, for example, by decomposing each of the different levels in a multigrid algorithm into subdomains or by constructing a dense linear system for the degrees of freedom on the boundaries between subdomains using a Schur complement [15]. Such approaches require multiple iterations between the local and nonlocal descriptions leading to multiple interprocessor communication steps.

Unfortunately, the trend in computer architectures is that processor speeds are increasing more rapidly than network speeds. For that reason, it is desirable to find algorithms with the smallest communications cost possible. One possible approach is to use the ideas behind the fast particle methods described above to construct a non-iterative domain decomposition method. One such method, based on the ideas in the FMM, was presented by Greengard and Lee in [10]. Greengard and Lee developed an adaptive grid-based Poisson solver of arbitrary accuracy. Their method divides the domain adaptively into square regions of various sizes and uses $K \times K$ Chebyshev polynomials to represent the charge and field on each region. These polynomials are accurate to order $O(H^K)$, where H is a local grid spacing on the adaptive mesh. The FMM is used to match the solutions on each patch, removing discontinuities at patch boundaries. The adaptivity of this method makes it difficult to estimate the amount of computation required for this method in all cases, but asymptotically, the method requires only $O(NK)$ work. Strain has developed another set of methods for the rapid solution of potential theory problems which is similar in spirit to the method of Greengard and Lee. In Strain's version, Fourier transforms are used to represent the charge and the field on each domain, and boundary conditions are matched by Ewald summation [16].

In this work, we present a non-iterative domain decomposition method for computing a finite difference approximation to the Poisson equation for the case of infinite domain boundary conditions. Parallelism is introduced by subdividing the discrete domain into patches. Poisson problems with infinite domain boundary conditions are solved independently on each of these patches, and a global coarse grid solution is used to communicate far-field effects to the patches. Each patch uses the global coarse solution, corrected with local fine grid information, to set

boundary conditions for a final fine grid solve. The first set of local solutions, as well as the global coarse solution, are based on Mehrstellen discretizations of the Laplacian [8], while the final set of local solutions are based on a standard five-point discretization. As was the case in [2], the discrete potential-theoretic properties of the Mehrstellen discretization play a critical role in providing a clean separation between the local and nonlocal contributions to the solution. Once the final solution on each patch is calculated, the union of these solutions represents an accurate fine grid solution to the global problem. Our method is completed in three large computation steps and two relatively small communication steps. Since our domain decomposition does not require repeated iteration between fine and coarse grids to converge toward a solution, the communication required on a parallel machine is significantly reduced.

This method has a number of other attractive properties. It is based on components that are routine to implement: multigrid on rectangular grids and N-body calculations that need not be "fast" (in the sense of [11]). It is also easily extended to three dimensions, using essentially the same components. We will discuss these and other possible extensions of these ideas in the final section of this paper.

2. ANALYSIS AND DISCRETIZATION OF THE INFINITE DOMAIN POISSON PROBLEM

We are interested in the solution of the Poisson equation on $\Re^2$ with a charge distribution ρ which has compact support. Specifically, we seek the solution ϕ to

$$\triangle\phi = \nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = \rho(x,y). \tag{1}$$

which has far-field behavior characterized by

$$\phi = -\frac{R}{2\pi}\log|\frac{1}{\vec{x}}| + o(1), \qquad |\vec{x}| \to \infty, \tag{2}$$

where R is the total charge:

$$R = \int_\Omega \rho(\vec{x})\, d\vec{x}, \tag{3}$$

and Ω contains the support of the charge ρ.

Using the maximum principle for harmonic functions, one easily shows that a solution to (1) and (2) is unique.

The infinite domain solution to the Poisson equation can be written as a convolution with the appropriate Green's function.

$$\phi(\vec{x}) = \int_\Omega \rho(\vec{y})G(\vec{y}-\vec{x})\, d\vec{y}, \tag{4}$$

where $G(\vec{z})$ is the Green's function:

$$G(\vec{z}) = -\frac{1}{2\pi}\log\left(\frac{1}{|\vec{z}|}\right). \tag{5}$$

A Boundary Integral Form of the Solution

We need to compute the infinite domain solution in terms of solutions on bounded domains. We do this using a technique that has been common in the plasma physics community for some time; see, e.g. [13] or [14].

Consider finding a solution

$$\phi^{DB} = \phi^D + \phi^B. \tag{6}$$

We define ϕ^D to be the solution of the Poisson equation with homogeneous Dirichlet boundary conditions:

$$\triangle \phi^D = \rho, \ \vec{x} \in \Omega, \tag{7}$$

$$\phi^D = 0, \ \vec{x} \in \partial\Omega, \text{ and} \tag{8}$$

$$\phi^D \equiv 0, \ \vec{x} \notin \Omega. \tag{9}$$

We define ϕ^B to be the logarithmic potential of a single layer:

$$\phi^B(\vec{x}) = \int_{\partial\Omega} \mu(\vec{y}(s)) G(\vec{x} - \vec{y}(s)) \, ds, \text{ where} \tag{10}$$

$$\mu(\vec{y}(s)) = \left. \frac{\partial \phi^D(\vec{y}(s))}{\partial n} \right|_{inside}. \tag{11}$$

The field due to a single layer on $\partial\Omega$ is harmonic both within Ω and outside of Ω. In addition, the potential is continuous as it crosses the boundary. However, the normal derivative of such a potential goes through a jump discontinuity at the boundary. Our choice of μ, then, is important: it is chosen such that ϕ^{DB} has a continuous normal derivative. That is,

$$\left[\frac{\partial \phi^D}{\partial n} \right] = - \left[\frac{\partial \phi^B}{\partial n} \right], \text{ and therefore} \tag{12}$$

$$\left[\frac{\partial \phi^{DB}}{\partial n} \right] = \left[\frac{\partial \phi^D}{\partial n} \right] + \left[\frac{\partial \phi^B}{\partial n} \right] = 0. \tag{13}$$

We find that our constructed solution meets our first criterion (1) by definition:

$$\triangle \phi^{DB} = \triangle(\phi^D + \phi^B) = \begin{cases} \rho, & \vec{x} \in \Omega \\ 0, & \vec{x} \notin \Omega \end{cases} \tag{14}$$

Furthermore, since ϕ^{DB} and $\frac{\partial \phi^{DB}}{\partial n}$ are continuous at $\partial\Omega$, it follows that

$$\triangle \phi^{DB}(\vec{x}) = 0, \ \vec{x} \in \partial\Omega. \tag{15}$$

To see that this solution satisfies the far-field condition, we need to use the divergence theorem and the multipole expansion of the Green's function. The divergence theorem states that for a piecewise smooth boundary $\partial\Omega$ and a continuously differentiable φ,

$$\int_{\partial\Omega} \frac{\partial \varphi}{\partial n} \, ds = \int_{\Omega} \triangle \varphi \, d\vec{y}. \tag{16}$$

Representing the Green's function by a multipole expansion, we have

$$\int_{\partial\Omega} \frac{\partial\varphi}{\partial n}(\vec{y}(s))G(\vec{x}-\vec{y}(s))\,ds = G(\vec{x})\int_{\partial\Omega}\frac{\partial\varphi}{\partial n}(\vec{y}(s))\,ds + \int_{\partial\Omega}\frac{\partial\varphi}{\partial n}O\left(\frac{|\vec{y}|}{|\vec{x}|}\right)ds \tag{17}$$

$$= G(\vec{x})\int_{\partial\Omega}\frac{\partial\varphi}{\partial n}(\vec{y}(s))\,ds + o(1). \tag{18}$$

Then substituting our definitions for ϕ^B and μ, we see that

$$\begin{aligned}\phi^B(\vec{x}) &= G(\vec{x})\int_{\partial\Omega}\frac{\partial\phi^D}{\partial n}(\vec{y}(s))\,ds + o(1)\\ &= G(\vec{x})\int_{\Omega}\triangle\phi^D\,d\vec{y} + o(1)\\ &= G(\vec{x})\int_{\Omega}\rho\,d\vec{y} + o(1)\end{aligned} \tag{19}$$

Since we have defined ϕ^D to be identically zero outside of Ω, in the far field we have

$$\phi^{DB} = \phi^D + \phi^B = \phi^B = G(\vec{x})\int_{\Omega}\rho\,d\vec{y} + o(1), \qquad |\vec{x}|\to\infty, \tag{20}$$

which satisfies our second criterion (2).

Finally, we note that it is possible to use this constructed solution as a boundary condition for another Dirichlet problem. That is, we can solve

$$\triangle\phi = \rho,\ \vec{x}\in\Omega^L,\ \text{and} \tag{21}$$

$$\phi = \phi^{DB},\ \vec{x}\in\partial\Omega^L,\ \text{where} \tag{22}$$

$$\Omega^L \supset \Omega. \tag{23}$$

Again, this is another way of representing the solution: ϕ and ϕ^{DB} are identically equal in Ω.

2.1. Discretization of the Infinite Domain Problem

We need a way to discretize the Poisson equation on a bounded domain and a way to discretize the boundary integral calculation. Given those two discretizations, a numerical solution to the infinite domain Poisson problem can be found by solving the Poisson equation with homogeneous Dirichlet boundary conditions for ϕ^D; calculating the boundary charge (11) on $\partial\Omega$ and evaluating the convolution integral (10) to set the far-field boundary conditions on $\partial\Omega^L$; and solving the Poisson equation a second time on Ω^L with the Dirichlet boundary conditions just set. We first discuss finite-difference discretizations of the Poisson equation and the properties of these discretizations. Then we briefly describe our discretization of the boundary integral calculation.

In order to calculate numerical solutions, we represent the solution on a finite discrete grid. We represent both the potential field, ϕ, and the charge, ρ, on a discrete, two-dimensional Cartesian grid. For convenience, our grid is equally spaced in both x and y directions, with grid spacing h. We index our grid by integer pairs $\vec{j} = (j_x, j_y)$. A point $\vec{j}$ on the grid corresponds to a point in the real plane $\vec{x}_{\vec{j}} = (j_x h, j_y h)$. The Cartesian grid just described is node-centered.

An exact solution, ϕ^{exact}, to the Poisson equation is represented on this discrete grid, such that

$$\phi^{exact,h}_{\vec{j}} = \phi^{exact}(\vec{x}_{\vec{j}}). \tag{24}$$

For our method we use two different discrete Laplacian operators: the standard five-point discretization,

$$(L_5\psi^h)_{\vec{j}} = \frac{\psi^h_{j_x,j_y+1} + \psi^h_{j_x,j_y-1} + \psi^h_{j_x+1,j_y} + \psi^h_{j_x-1,j_y} - 4\psi^h_{\vec{j}}}{h^2}, \tag{25}$$

and the nine-point discretization,

$$\begin{aligned}(L_9\psi^h)_{\vec{j}} \;=\; & \frac{1}{6h^2}\Big(\psi^h_{j_x+1,j_y+1} + \psi^h_{j_x+1,j_y-1} + \psi^h_{j_x-1,j_y+1} + \psi^h_{j_x-1,j_y-1} \\ & + 4(\psi^h_{j_x,j_y+1} + \psi^h_{j_x,j_y-1} + \psi^h_{j_x+1,j_y} + \psi^h_{j_x-1,j_y}) \\ & -20\psi^h_{\vec{j}}\Big).\end{aligned} \tag{26}$$

Both of these Laplacian operators are accurate to $O(h^2)$, but the truncation error of the L_9 operator takes a special form. If $\phi^{exact,h}$ is the exact solution evaluated at grid points, and the truncation error, $\tau^h_{\vec{j}}$, is defined as

$$\tau^h_{\vec{j}} = \rho^h_{\vec{j}} - (L_9\phi^{exact,h})_{\vec{j}}, \tag{27}$$

we can use Taylor expansion, along with the fact that $\triangle^2\phi = \triangle\rho$, to determine that

$$\begin{aligned}\tau^h_{\vec{j}} \;&=\; \rho^h_{\vec{j}} - (L_9\phi^{exact,h})_{\vec{j}} \\ &=\; -\frac{h^2}{12}\triangle\rho - \frac{h^4}{360}\left(\frac{\partial^4\rho}{\partial x^4} + 4\frac{\partial^4\rho}{\partial x^2\partial y^2} + \frac{\partial^4\rho}{\partial y^4}\right)\Big|_{\vec{x}_{\vec{j}}} + O(h^6).\end{aligned} \tag{28}$$

By preconditioning the charge and solving a slightly different equation:

$$L_9\phi^{*,h} = \rho^{*,h} = \rho^h + \frac{h^2}{12}L_5\rho^h, \tag{29}$$

the truncation error is reduced to $O(h^4)$. Thus by combining the nine-point Laplacian operator with suitable preconditioning of the right hand side, we can construct a fourth-order accurate method. Such methods are called Mehrstellen methods [8]. Note that in the special case of a harmonic function, where $\triangle\phi^e = 0$, preconditioning is unnecessary, the first two terms in the truncation error vanish, and the truncation error (28) associated with the L_9 operator is reduced to $O(h^6)$.

It is important to our method to understand not only the behavior of the truncation error but also the nature of the solution error in the far field. Specifically, we find that the solution error in the far field differs from a harmonic function by $O(h^6)$. To see this, first note that the computed ϕ is the solution to

$$\phi = (L_9)^{-1}\rho. \tag{30}$$

We can analyze the inverse operator, $(L_9)^{-1}$, without actually constructing it. We know

$$\triangle^{-1}\rho = \phi^e = L_9^{-1}\rho + L_9^{-1}(\mathcal{L}(\rho)) + O(h^6), \tag{31}$$

where the operator $\mathcal{L}$ is defined as

$$\mathcal{L}(\psi) \;=\; \frac{h^2}{12}(\triangle\psi) + \frac{h^4}{360}\left(\frac{\partial^4\psi}{\partial x^4} + 4\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4}\right). \tag{32}$$

This equation can be rearranged and applied recursively to find that

$$\phi = \phi^e - \triangle^{-1}\mathcal{L}(\rho) + \triangle^{-1}\mathcal{L}(\mathcal{L}(\rho)) + O(h^6). \tag{33}$$

Away from the support of ρ, this relationship can be represented as

$$\phi|_{\sim\mathrm{supp}(\rho)} = \phi^e + h^4\mathcal{H} + O(h^6), \tag{34}$$

where $\mathcal{H}$ is a harmonic function, $\triangle\mathcal{H} = 0$.

This result represents two significant characteristics of the discrete solution away from the support of ρ. First, the error in the discrete solution is a local function of ρ, and therefore we can expect our solution to be accurate to $O(h^4)$ away from the support of ρ. Second, the leading-order term in the error is a harmonic function in that region. We emphasize that both these properties hold without the use of the preconditioning described in (29). These are distinctive characteristics of the nine-point Mehrstellen Laplacian discretization.

We choose to solve the bounded, discrete Poisson problems that arise in this method with a variant of multigrid. Since our domain sizes are often not powers of 2, not all grids can be evenly coarsened. When a grid cannot be evenly coarsened, we increase the size of the grid by one, padding the residual field with zeros. This growing multigrid method is quite robust when W-cycles are used, reducing the residual by a factor of 10^{10} in 8 to 17 iterations. It should be noted that the method used to accelerate convergence of the solution is not essential to our algorithm: any fast solver would suffice.

While the specific acceleration method is not critical, the differences between the five- and nine-point Laplacian operators are very important, when implementing our domain decomposition method, which follows.

With a discretized representation of ϕ^D, it is also possible to evaluate ϕ^B numerically. We can obtain an estimate of the boundary charge, μ, of (11) to the required accuracy by a one-sided difference approximation. For example, for a point $\vec{j}$ on the right-hand boundary of Ω^h,

$$\mu_{\vec{j}} \;=\; \frac{1}{h}\left(\frac{25}{12}\phi^D_{\vec{j}} - 4\phi^D_{j_x-1,j_y} + 3\phi^D_{j_x-2,j_y}\right.$$

$$-\frac{4}{3}\phi^D_{j_x-3,j_y} + \frac{1}{4}\phi^D_{j_x-4,j_y}\Bigg)$$
$$= \frac{\partial \phi^D}{\partial x}(\vec{x}_j) + O(h^4). \tag{35}$$

We evaluate the integral (10) numerically, using Simpson's rule.

The algorithm we use to find a discrete solution to the infinite domain Poisson problem on a single grid can now be be summarized in three steps:

- solve the initial Poisson problem with Dirichlet boundary conditions:

$$\begin{aligned} L\phi^{h,initial} &= \rho^h, \qquad \vec{x} \in \Omega^{h,initial}, \\ \phi^{h,initial} &= 0, \qquad \vec{x} \in \partial\Omega^{h,initial}; \end{aligned} \tag{36}$$

- calculate the boundary charge and integrate around the boundary to set the far-field boundary conditions as in (10) and (35), setting $\phi^D = \phi^{h,initial}$ in (35) and using Simpson's rule to calculate $\phi^h = \phi^B$ of (10) for all $\vec{x} \in \partial\Omega^h$;
- solve a Poisson problem with the Dirichlet boundary conditions just set:

$$\begin{aligned} L\phi^h &= \rho^h, \qquad \vec{x} \in \Omega^h, \\ \phi^h &= \phi^B(\vec{x}), \qquad \vec{x} \in \partial\Omega^h. \end{aligned} \tag{37}$$

Using the analysis given above, we find that

$$\phi^h = \phi^e + \frac{h^2}{12}\rho + h^4\mathcal{H} + O(h^6), \tag{38}$$

where $\mathcal{H}$ is a function that is harmonic away from the support of ρ and includes contributions from the truncation error and the error in the boundary conditions.

3. DOMAIN DECOMPOSITION

We would like an $O(h^2)$ solution to

$$\triangle\phi = \rho \tag{39}$$

with infinite domain boundary conditions. We have a method to compute the solution on a discrete domain Ω^h, but we would like to subdivide the problem so that we can solve the problem in parallel.

Let us define a domain decomposition for the problem. Given a domain Ω which contains the support of ρ, we divide Ω into L patches, Ω_l, such that

$$\Omega = \bigcup_{l=1}^{L} \Omega_l, \tag{40}$$

$$\rho = \sum_l \rho^l, \text{ and} \tag{41}$$

$$\text{supp}(\rho^l) \subset \Omega_l. \tag{42}$$

Then we seek a solution

$$\begin{aligned}\phi(\vec{x}) &= (\Delta^{-1}\rho)(\vec{x}) = \sum_l (\Delta^{-1}\rho^l)(\vec{x}) = \sum_l \phi^l(\vec{x}) \\ &= \sum_{l\,:\,\Omega_l \text{ near } \vec{x}} \phi^l(\vec{x}) + \sum_{l\,:\,\Omega_l \text{ far from } \vec{x}} \phi^l(\vec{x}). \end{aligned} \tag{43}$$

The calculation of the near field of ϕ^l involves only local work and local communication. The first sum in (43) then involves local work, proportional to the size of the support of ρ^l, and the calculations are independent for each l. The second sum in (43) represents the effect of far-field harmonic functions, which are very smooth, in fact real analytic functions, and therefore can be represented with a much smaller number of computational degrees of freedom. For particles, the Fast Multipole Method uses multipole expansions to obtain a compact representation of the far field, while the Method of Local Corrections uses a representation based on a finite difference calculation on a coarse grid using a Mehrstellen discretization. We construct a version of the Method of Local Corrections for finite difference problems.

Before we describe our algorithm fully, we need to define some of the notation and operators we will be using. First, we define a rectangular domain on a Cartesian grid by specifying the integer indices of the lower left and upper right corners, $\vec{l}$ and $\vec{u}$. Given an $\Omega^h = [\vec{l}, \vec{u}]$ with grid spacing h, we can define various operators which return modified domains. Since we are interested in grids which can be indexed by integers, we take care that all of our operators return grids which also can be indexed by integers.

We define a coarsening operator, *Coarsen*, such that, given $\Omega^h = [\vec{l}, \vec{u}]$,

$$Coarsen(\Omega^h, N_r) = \Omega^H = [\lfloor \frac{\vec{l}}{N_r} \rfloor, \lceil \frac{\vec{u}}{N_r} \rceil], \text{ where } H = N_r h. \tag{44}$$

We also find it useful to define an operator, *Grow*, which returns a larger or smaller domain growing the given domain by a certain amount in each direction. Given $\Omega^h = [(\mathtt{l}_x, \mathtt{l}_y), (\mathtt{u}_x, \mathtt{u}_y)]$, *Grow* can be defined as

$$Grow(\Omega^h, N) = \Omega^{h,g} = [\vec{l} - (N, N), \vec{u} + (N, N)]. \tag{45}$$

Note that *Grow* is well-defined even with negative integers N, in which case the returned domain is smaller, rather than larger, than the original domain.

Finally, we need a sampling operator to extract coarse grid representations from fine grid data:

$$\psi^H_{\vec{j}} = (Sample^H(\psi^h))_{\vec{j}} = (\psi^h)_{N_r \vec{j}}, \text{, where } H = N_r h \tag{46}$$

Sampling is possible since we are using node-centered grids; a more elaborate averaging operator would be required for a cell-centered method.

3.1. Algorithm Description

Let us consider a fine grid discretization Ω^h, corresponding to the domain Ω, as in the previous section. We decompose Ω^h into L grids with disjoint interiors such

that they intersect only at their boundaries and that

$$\Omega^h = \bigcup_{l=1}^{L} \Omega_l^h. \tag{47}$$

Associated with each Ω_l^h are two other grids. First we have a larger fine grid, $\Omega_l^{h,g} = \mathit{Grow}(\Omega_l^h, N_r D)$. Because $\Omega_l^{h,g}$ is defined on a larger domain than Ω_l^h, the various $\Omega_l^{h,g}$ overlap. The degree of overlap is a function of both a correction radius, D, and the refinement ratio, N_r. We also define a coarse grid $\Omega_l^H = \mathit{Coarsen}(\Omega_l^{h,g}, N_r)$ representing a projection of the grid $\Omega_l^{h,g}$ onto a coarser mesh with grid spacing $H = N_r h$.

Finally, we define a coarse grid corresponding with the entire problem domain: $\Omega^H = \mathit{Grow}(\mathit{Coarsen}(\Omega^h, N_r), D)$. Our algorithm can be described generally in terms of this collection of discrete grids.

We seek to approximate the solution to

$$L_5 \phi^{single,h} = \rho^h \tag{48}$$

where $\phi^{single,h}$ is the discrete solution on a single grid covering domain Ω^h, with grid spacing h. As shown previously, L_5 is a second-order accurate approximation to the Laplacian operator, $\triangle$, and $\phi^{single,h}$ is a second-order approximation to the exact solution of the Poisson equation, ϕ^e.

We define a discrete solution ϕ^h on the domain decomposition described in (47), representing this solution as $\phi^{h,l}$ on each subdomain Ω_l^h . We want to compute ϕ^h such that

$$\phi^h = \phi^{single,h} + O(h^2). \tag{49}$$

In fact, we hope that

$$\|\phi^h - \phi^{single,h}\| << \|\phi^{single,h} - \phi^{exact,h}\|, \tag{50}$$

where $\phi^{exact,h}$ is the exact solution evaluated at grid points.

Our algorithm calculates the solution, $\phi^{h,l}$, on each Ω_l^h in three main steps.

- Initial Local Solve. On each local patch, $\Omega_l^{h,g}$, solve

$$L_9 \phi^{h,l,initial} = \rho^{h,l} \tag{51}$$

with infinite domain boundary conditions, and construct a coarse representation of this data:

$$\phi^{H,l,initial} = \mathit{Sample}^H(\phi^{h,l,initial}) \tag{52}$$

- Global Coarse Grid Solve. Compute a single coarse-grid solution to communicate global information

$$L_9 \phi^H = R^H, \text{ on } \Omega^H, \tag{53}$$

with infinite domain boundary conditions, and with R^H defined by

$$R^H = \sum_l R^{H,l}, \text{ and} \tag{54}$$

$$R^{H,l} = \begin{cases} L_9\phi^{H,l,initial}, & \text{on } Grow(\Omega_l^H, -1) \\ 0, & \text{otherwise.} \end{cases} \tag{55}$$

- FINAL LOCAL SOLVE. On each local patch Ω_l^h, solve

$$L_5\phi^{h,l} = \rho^{h,l}, \tag{56}$$

with boundary conditions set by a combination of data from nearby patches and data from the coarse grid which has been adjusted to remove local effects:

$$\begin{aligned}(\phi^{h,l})_{\vec{j}} \;=\; & \sum_{l':\vec{j}\in\Omega_{l'}^{h,g}} \phi_{\vec{j}}^{h,l',initial} \\ & + \mathcal{I}(\phi^H - \sum_{l':\vec{j}\in\Omega_{l'}^{h,g}} \phi^{H,l',initial})_{\vec{j}},\ \vec{j} \in \partial\Omega_l^h,\end{aligned} \tag{57}$$

where $\mathcal{I}$ is an interpolation operator used to transfer information from the coarse grid to the fine grid.

The first two steps are completely defined in the previous discussion, but one part of this algorithm deserves special attention: the calculation of the fine grid boundary condition for the final local solve.

Setting Fine Grid Boundary Conditions for the Final Solve

To set the boundary of a patch, we use both data from the local fine solutions and interpolants from the large coarse solution. We must be careful always to use the local data where it is available and yet not to add information twice by leaving the local effects in the coarse grid data which we interpolate.

The easiest way to keep track of all this is with a stencil. A sample interpolation stencil is shown in Figure 1. We move the stencil around the sides of each patch such that its center row covers the projection of the boundary points which we wish to set. Coarse grid data is interpolated from the nine coarse grid points which lie under the stencil. Note that for accurate interpolation we need consistent data from all nine points. In Figure 1, for example, we would subtract $\phi^{h,m,initial}$ from the coarse grid stencil values, but we would not subtract $\phi^{h,n,initial}$, since the stencil is not contained entirely by Ω_n^H. We later add local data from $\phi^{h,m,initial}$, but not from $\phi^{h,n,initial}$, since those values were included in the coarse grid interpolant.

Figure 2 shows the fine grid points we interpolate from the coarse grid stencil. Note that stencils generally share the fine points furthest from the center with neighboring stencils. When multiple stencils can be used to evaluate a fine grid point, we average all the values produced by the various stencils.

We are able to construct an interpolant which is accurate to $O(H^5)$ from our coarse stencil. Since we have subtracted all local information from our stencil

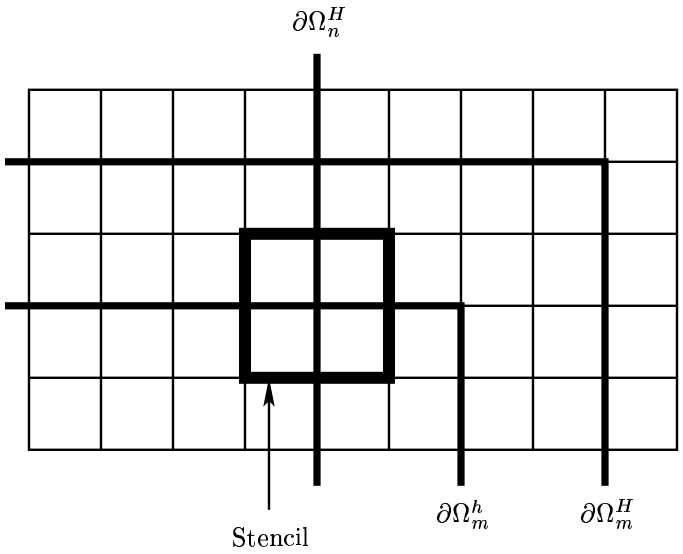

FIG. 1. An interpolation stencil as placed on the coarse grid.

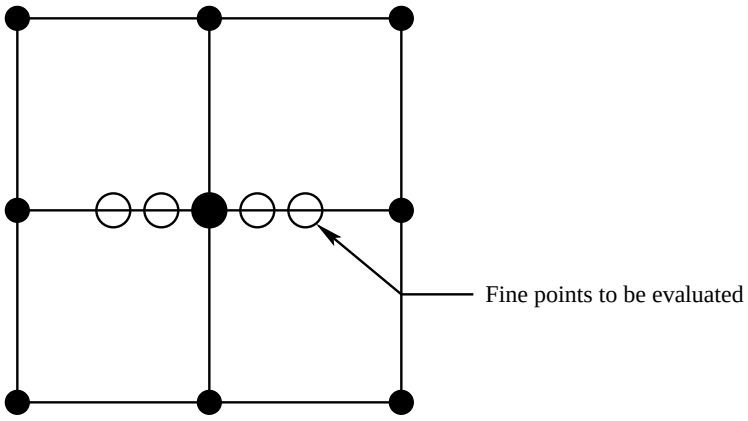

FIG. 2. An interpolation stencil as used to evaluate fine grid values.

values:

$$\phi^{H,stencil} = \phi^H - \sum_{m:\Omega_m^H \supset \Omega^{H,stencil}} \phi^{H,m,initial}; \tag{58}$$

we have also made the coarse stencil data harmonic:

$$L_9\phi^{H,stencil} = L_9\phi^H - \sum_{m:\Omega_m^H \supset \Omega^{H,stencil}} L_9\phi^{H,m,initial} = 0. \tag{59}$$

We interpolate by constructing approximate derivatives and a Taylor series representation of a local solution. Since the coarse stencil values are harmonic, we are able to use a compact, 3×3 stencil to compute sufficiently accurate approximations of the derivatives necessary as interpolation coefficients to guarantee that the interpolated results, $\phi^{h,stencil}$, are accurate to $O(H^5)$ overall.

Let us consider, for example, a top or bottom boundary, for which the fine grid values of j_x vary but j_y remains constant. The difference approximations to the necessary derivatives are

$$\begin{aligned}
\frac{\partial^4\psi}{\partial x^4} &= -\frac{\partial^4\psi}{\partial x^2\partial y^2} \\
&= -\frac{1}{H^2}(4\psi^H_{\vec{j}} - 2(\psi^H_{j_x+1,j_y} + \psi^H_{j_x-1,j_y} + \psi^H_{j_x,j_y+1} + \psi^H_{j_x,j_y-1}) \\
&\qquad + \psi^H_{j_x+1,j_y+1} + \psi^H_{j_x+1,j_y-1} + \psi^H_{j_x-1,j_y+1} + \psi^H_{j_x-1,j_y-1}) \\
&\qquad + O(H^2), \qquad (60) \\
\frac{\partial^3\psi}{\partial x^3} &= -\frac{\partial^3\psi}{\partial x\partial y^2} \\
&= \frac{1}{2H^3}(\psi^H_{j_x-1,j_y+1} + \psi^H_{j_x-1,j_y-1} - \psi^H_{j_x+1,j_y+1} - \psi^H_{j_x+1,j_y-1} \\
&\qquad + 2(\psi^H_{j_x+1,j_y} - \psi^H_{j_x-1,j_y})) + O(H^2), \qquad (61) \\
\frac{\partial^2\psi}{\partial x^2} &= \frac{1}{H^2}(\psi^H_{j_x+1,j_y} - 2\psi^H_{\vec{j}} + \psi^H_{j_x-1,j_y}) - \frac{H^2}{12}\frac{\partial^4\psi}{\partial x^4} + O(H^4), \text{ and} \qquad (62) \\
\frac{\partial\psi}{\partial x} &= \frac{1}{2H}(\psi^H_{j_x+1,j_y} - \psi^H_{j_x-1,j_y}) - \frac{H^2}{6}\frac{\partial^3\psi}{\partial x^3} + O(H^4), \qquad (63)
\end{aligned}$$

where the values of $\vec{j}$ refer to the indices of the coarse stencil. Note that the approximation (60) is needed for (62), and (61) is needed for (63). We use these approximations in a Taylor series approximation to interpolate fine grid values:

$$\phi^h_{\vec{j}+\vec{j}_h} = \mathcal{I}(\phi^H)_{\vec{j}} = \phi^H_{\vec{j}} + \delta_h\frac{\partial\phi}{\partial x} + \frac{\delta_h^2}{2}\frac{\partial^2\phi}{\partial x^2} + \frac{\delta_h^3}{6}\frac{\partial^3\phi}{\partial x^3} + \frac{\delta_h^4}{24}\frac{\partial^4\phi}{\partial x^4} + O(H^5). \tag{64}$$

Here we still have $\vec{j}$ as the coarse grid index. The index $\vec{j}_h$ represents the displacement, in fine grid spacing, from the projection of the center of the coarse stencil onto the fine grid. The distance, δ_h, is the quantity $h|\vec{j}_h|$.

Finally, we add back the local fine grid information to the interpolated data in the stencil:

$$\phi^{h,stencil} = \phi^{h,stencil} + \sum_{m:\Omega_m^{h,g} \supset \Omega^{H,stencil}} \phi^{h,m,initial}. \tag{65}$$

We copy these values into the appropriate boundary cells of $\phi^{h,l}$.

Once we have traversed the boundaries of every patch and set the boundary conditions, we use multigrid to solve

$$L_5\phi^{h,l} = \rho^{h,l}. \tag{66}$$

Our solution, ϕ^h, is then a composite of the $\phi^{h,l}$ calculated on each subdomain. Since our boundary condition sets coincident boundaries exactly the same, we do not need to worry about ϕ^h being multi-valued at the intersections of subdomains.

We will later demonstrate that the combination of fine and coarse data which we used to set boundary conditions is accurate to $O(h^2) + O(H^4)$ anywhere we wish to apply it. We need not set boundary values along the original subdomain boundaries: we could divide the original domain into a different set of patches, use the interpolation and correction as described to set the boundary conditions on the new patches, and expect the final solution to be just as accurate. Indeed, we could achieve the same overall accuracy by applying the stencil everywhere to generate fine grid values.

In many applications, our approach of setting boundary conditions for a final solve is preferable. With such a method, the error away from the boundaries of the subdomains is sufficiently smooth that one can apply difference operators to the solution to approximate, e.g., $\nabla\phi$, to $O(h^2)$ accuracy.

3.2. Error Analysis of MLC

The algorithm we have just presented divides a charge field ρ with support confined to Ω into several small charge fields ρ^l defined on subdomains Ω_l. We can understand most of the properties of this algorithm by looking at a simpler subproblem, such that the right hand side, ρ, is a charge with bounded support on some smaller domain, Ω_ρ.

Let us consider the application of our algorithm to this simpler problem. We subdivide the problem into patches such that a single patch, Ω_0^h, contains the support of ρ. Our solution is found in three steps.

- First, solve

$$L_9\phi^{h,0,initial} = \rho^{h,0} \tag{67}$$

on $\Omega_0^{h,g}$ with infinite domain boundary conditions.

- Next, solve

$$L_9\phi^H = R^H \tag{68}$$

on Ω^H, the coarse domain covering the entire problem, with infinite domain boundary conditions.

- Finally, set boundary conditions on each patch as follows:

$$\phi^{h,l} = \begin{cases} \phi^{h,0,initial} + \mathcal{I}(\phi^H - \phi^{H,0,initial}), & \text{for boundary points in } \Omega_0^{h,g} \\ \mathcal{I}(\phi^H), & \text{otherwise,} \end{cases} \tag{69}$$

and solve a Poisson equation on each patch with the Dirichlet boundary conditions just set.

In order to better understand our result, let us construct an auxiliary solution, the solution to

$$L_9\bar{\phi}^h = \rho^h \tag{70}$$

on Ω^h with infinite domain boundary conditions.

We have shown previously that $\bar{\phi}^h$ and $\phi^{h,0,initial}$ are accurate to $O(h^2)$ on their domains. If we can show that the boundary conditions we set, (69), are accurate to $O(h^2)$ as well, it follows that our algorithm is accurate to $O(h^2)$ overall. It is sufficient to show

$$\bar{\phi}^h - [\phi^{h,0,initial} - \mathcal{I}(\phi^H - \phi^{H,0,initial})] = O(h^2), \tag{71}$$

for all boundary points in $\Omega_0^{h,g}$, and

$$\bar{\phi}^h - \mathcal{I}(\phi^H) = O(h^2), \text{ otherwise.} \tag{72}$$

First, let us note that while $\bar{\phi}^h$ and $\phi^{h,0,initial}$ are both only accurate to $O(h^2)$, by (38), they differ from one another by $O(h^4)$, i.e.

$$\bar{\phi}^h - \phi^{h,0,initial} = O(h^4). \tag{73}$$

Let us now define

$$\tilde{R}^H = \begin{cases} L_9(Sample^H(\bar{\phi}^h)), & \text{outside } \Omega_0^H \\ 0, & \text{in } \Omega_0^H. \end{cases} \tag{74}$$

$\tilde{R}^H$ represents the error we make in our method by truncating $\tilde{R}^H$ to be zero outside the correction radius. Note that

$$\phi^H = \bar{\phi}^h + (L_9)^{-1}\tilde{R}^H + \xi^B, \tag{75}$$

where ξ^B represents the error in the infinite domain boundary condition, which is $O(H^4)$. Since $\bar{\phi}^h$ differs from a harmonic function by $O(h^6)$ outside $\Omega_0^{h,g}$ (away from the support of ρ), we know

$$L_9(Sample^H(\bar{\phi}^h)) = O(H^6) \tag{76}$$

outside Ω_0^H, and therefore

$$\tilde{R}^H = O(H^6). \tag{77}$$

Thus (75) becomes

$$\phi^H = Sample^H(\bar{\phi}^h) + O(H^4), \tag{78}$$

and the resulting error estimate for boundary points within $\Omega_0^{h,g}$ is

$$\xi^{h,BC} = O(h^4) + O(H^4). \tag{79}$$

For points outside $\Omega_0^{h,g}$, we note again that

$$\phi^H = \bar{\phi}^h + O(H^4), \tag{80}$$

and that $\bar{\phi}^h$ is harmonic in this region. Therefore ϕ^H differs from a harmonic function by terms that are $O(H^4)$, and the far-field interpolated values obtained in (69) differ from $\bar{\phi}^h$ by $O(H^4)$.

The resulting composite solution, then, is accurate to $O(H^4) + O(h^2)$. As long as the refinement ratio is chosen correctly, such that the $O(H^4)$ and $O(h^2)$ terms are comparable, we can guarantee an accuracy of $O(h^2)$ over the entire domain Ω^h.

3.3. Parallel Implementation

The implementation of this algorithm in parallel is fairly straightforward. Given a parallel machine with N_{procs} processors we assign patches associated with the fine grids of the domain decomposition cyclically to each of the processors. Generally we use N_{procs} equal to exactly the number of patches in the decomposition. Both the initial local infinite domain Poisson solves and the final multigrid solves on the fine grids can be performed in parallel without any communication among the processors during those phases of the computation.

In a fashion similar to Baden's parallel implementation of the MLC [4], we have chosen to construct the global coarse grid representation of both the right hand side and the solution on each processor. Since the global coarse grid requires fewer grid points than any individual fine grid, this overhead is small both in terms of memory and computation required. In order to build the global coarse representation, all the processors need to broadcast the coarse grid right hand sides for each of their patches to all the other processors. This communication step creates the global coarse right hand side necessary for the global infinite domain Poisson solve. All the processors then perform that same solve, after which every processor has a copy of the coarse grid field necessary for the final phase of the computation.

The only other communication necessary is the communication among patches to correct the interpolation of the fine grid boundary conditions. This communication occurs between processors which control neighboring patches and only points along the boundary are communicated, so the communication required for this step is rather small.

3.4. The Relationship among Patches, Refinement Ratio, and Problem Size

We would like for our algorithm to be scalable on a parallel machine. To ensure scalability, we need to consider the effect of the parts of the algorithm which are performed as serial rather than parallel computations. In our algorithm, the global coarse grid infinite domain solve is the only serial calculation. We need to show that this part of the calculation can be scaled such that it does not limit the parallel performance of the algorithm as a whole.

Let us consider a problem domain represented on a fine grid of dimensions $[0, N] \times [0, N]$. We divide the domain into N_p patches in each direction for a total of $N_p \times N_p$ patches. We represent the global coarse data on a grid of dimensions $[0, N^C] \times [0, N^C]$. The ratio of coarse grid spacing to fine grid spacing is the refinement ratio $N_r = \frac{N}{N^C}$.

Scalability requires that the work done on the global coarse solve be proportional to the work done on each patch. The work done on the global coarse solve is proportional to the size of the coarse grid: $O((N^C)^2) = O(\frac{N^2}{N_r^2})$. The work per patch is $O(\frac{N^2}{N_p^2})$. This implies that we need a refinement ratio N_r proportional to the number of patches in each direction N_p.

Now we must consider whether it is feasible to increase the refinement ratio as we increase the number of patches. We expect our global solve to be accurate to $O(H^4)$ while the overall method is only accurate to $O(h^2)$. This implies a relationship between the refinement ratio and size of the problem since $N_r = \frac{H}{h}$. We need

$$(N_r h)^4 \propto h^2, \tag{81}$$

$$N_r^4 \propto \frac{1}{h^2}, \text{ or} \tag{82}$$

$$N_r^2 \propto N. \tag{83}$$

Combining this relationship with the relationship between patches and refinement ratio, we have

$$N_r^2 \propto N_p^2 \propto N. \tag{84}$$

If we hold to these relationships, the algorithm should exhibit both good parallel performance and $O(h^2)$ accuracy overall.

3.5. Computational Overhead

In order to analyze the conditions under which our method is most attractive, we need to estimate the overhead incurred in performing this type of domain decomposition. Multigrid typically requires $O(M \log M)$ work to calculate a solution to Poisson's equation on a grid of M points. For the purposes of this analysis, we will refer to this amount of work as M work units, ignoring the log term. Note that this simplification favors large single grids over domains decomposed into smaller grids.

First, let us consider the single grid infinite domain problem. The solution for a domain $\Omega^{h,0}$ requires first a multigrid solution on an intermediate domain, $\Omega^{h,initial}$, and then a solution on a larger domain, Ω^h. For good accuracy we need Ω^h to be at least 10% larger than $\Omega^{h,0}$ in each direction. This requirement results in an overall size of $1.2N \times 1.2N = 1.44N^2$. The size of $\Omega^{h,initial}$, is then $1.1N \times 1.1N = 1.21N^2$. The two multigrid solutions combined require $1.44N^2 + 1.21N^2$ work units. The potential calculation also requires $O(N^2)$ work, but the cost is small compared to the multigrid solutions, and we will not consider this portion of the work. The work required for the single grid solution is then at least $2.65N^2$ work units.

For the parallel solution, our method involves three large steps: first we solve a number of independent problems with infinite domain boundary conditions; then, after aggregating coarse data, we solve the coarse grid problem for the entire domain with infinite domain boundary conditions; and finally, after setting accurate boundary conditions, we solve local fine grid problems with Dirichlet boundary

conditions. Communication steps are required between the major computational steps, but we are interested in estimating only the computational overhead here.

For a problem of size $N \times N$, we use $N_p \times N_p$ processors and a refinement ratio of N_r to determine the coarse grid spacing. We require $N_p^2 \propto N_r^2 \propto N$ for accuracy reasons, so let us take

$$N_p = a\sqrt{N}, \text{ and} \tag{85}$$
$$N_r = b\sqrt{N}. \tag{86}$$

For the first step of our algorithm, we solve infinite domain boundary condition problems of size $(\frac{N}{N_p} + 4N_r)^2$ on each of our N_p^2 independent fine grids. These solutions, as in the single grid case, require two separate multigrid solutions. The first set of solutions, on intermediate grids of size $(\frac{N}{N_p} + 2N_r)^2$, requires

$$W_{1a} = N_p^2 \left(\frac{N}{N_p} + 2N_r\right)^2 \tag{87}$$
$$= a^2 N \left(\frac{\sqrt{N}}{a} + 2b\sqrt{N}\right)^2 \tag{88}$$
$$= (1 + 4ab + 4(ab)^2)N^2 \text{ work units,} \tag{89}$$

and the second set of solutions requires

$$W_{1b} = N_p^2 \left(\frac{N}{N_p} + 4N_r\right)^2 \tag{90}$$
$$= a^2 N \left(\frac{\sqrt{N}}{a} + 4b\sqrt{N}\right)^2 \tag{91}$$
$$= (1 + 8ab + 16(ab)^2)N^2 \text{ work units.} \tag{92}$$

Taken together, again ignoring the cost of the potential calculation, the first step in our method requires

$$W_1 = (2 + 12ab + 20(ab)^2)N^2 \text{ work units.} \tag{93}$$

It is important to note, however, that the work required for this step can not be less than $2.65N^2$ work units: the individual infinite domain solutions are subject to the same constraints on domain sizes as in the single grid solution, and thus our method requires at least as much work as the single grid solve.

We have chosen to implement the second step of our algorithm, the global coarse grid solution with infinite domain boundary conditions, as a calculation which is replicated across all the processors. Thus our $N_p^2 = a^2 N$ processors each solve a problem of size $(\frac{N}{N_r})^2 = \frac{N}{b^2}$. These problems have the same overhead as a single grid calculation, so we can estimate the work for this second stage as

$$W_2 = 2.65\frac{a^2}{b^2}N^2 \text{ work units.} \tag{94}$$

Finally, we need to solve the independent problems on each domain with Dirichlet boundary conditions. The work required for this step is simply

$$W_3 = N^2 \text{ work units.} \tag{95}$$

Taken together, we see that the total work must be at least $3.65N^2$ work units. We can ensure that the work is not much greater than that, however. Choosing N_r and N_p such that $ba \leq \frac{1}{16}$ is sufficient to ensure that $W_1 < 2.83N^2$ work units. Choosing N_r and N_p such that $\frac{b}{a} \geq 4$ ensures that $W_2 \leq 0.17N^2$ work units. Together, these conditions ensure that the total work is less than $4N^2$ work units.

In terms of N_p, N_r, and N, the conditions for small computational overhead are expressed as

$$N_r \leq \frac{1}{2}\sqrt{N}, \text{ and} \tag{96}$$

$$N_p \leq \frac{1}{4}N_r. \tag{97}$$

These conditions imply that the smallest problem we can solve with this little overhead is a 65×65 problem on a single patch with $N_r = 4$. The smallest problem we can solve in parallel with such a small amount of computational overhead is a 257×257 problem on 2×2 patches with $N_r = 8$. Using these estimates to calculate theoretical speed-up, with a 7% overhead of serial computation for the global coarse grid solution, we would expect our solution of the 257×257 problem on four processors to be 2.65 times as fast as the single grid calculation.

These constraints do not seem excessive since the algorithm is specifically designed for solving large problems on large parallel machines and since we have removed almost all the communication overhead required by a standard multigrid method.

4. RESULTS

We have implemented versions of our algorithm in order to test our claims. Computer programs were written in two separate programming environments designed for building parallel algorithms for scientific computations: KeLP and Titanium. KeLP, designed and written by Baden, Fink, and Kohn [5] [9], is a C++ library that provides a large number of useful features both for parallel programming and for scientific computing. KeLP provides for interfaces between high-level C++ descriptions of the data structures and lower-level numerical algorithms, implemented in Fortran. This division allows the user to take advantage of the performance provided by Fortran compilers as well as the abstractions supplied by the C++ language. Titanium [17] is a dialect of Java developed for parallel scientific applications. The language has a SPMD model of parallel execution and has built-in abstractions which are useful for scientific applications. The Titanium compiler aggressively optimizes many of the most common routines in scientific computations, such as loops over two or three dimensional arrays.

The performance of the Titanium code is slightly slower than the KeLP implementation for our algorithm, but since the Titanium code was more easy to write

and modify, it proved more amenable to algorithm development and testing. All the results presented here are generated by code written in Titanium. Timing tests were run on the IBM SP2 at the San Diego Supercomputing Center. Additional results and comparisons including data for the KeLP implementation can be found in [7].

One of the benefits of solving the Poisson equation is that we can easily create test problems for which exact analytical solutions can be calculated directly. In such cases, we are able to measure the error in our computed solution by comparing it with the exact solution. We also use test problems for which the exact solution is not known. In these cases we perform Richardson error estimation, using a series of increasingly refined solutions, comparing each to the next finer. In all cases, we can calculate a convergence rate for our method from norms of the error estimates as the grid is refined.

There are several properties that we seek to demonstrate about our algorithm. First, we show that our algorithm is accurate to $O(h^2)$. Specifically, we show that the solution converges as predicted when the problem is scaled appropriately, i.e. when the refinement ratio, N_r, and the number of patches (in each direction), N_p, are inversely proportional to the square root of the grid spacing, h. In addition, we show that our algorithm produces more accurate results than a single infinite domain boundary condition solution on coarse grid data with Mehrstellen preconditioning (29). Finally, we measure the performance of the algorithm on a parallel machine, testing communication costs and speed-up.

4.1. The Test Problems

We have two similar test problems for which we can construct solutions. Both problems have radially symmetric charge distributions. The first charge distribution is very smooth:

$$\rho = \begin{cases} (\frac{r}{R_0} - \frac{r^2}{R_0^2})^4, & r < R_0 \\ 0, & r \geq R_0. \end{cases} \tag{98}$$

This charge can be integrated directly to produce an exact solution:

$$\phi = \begin{cases} r^2 \left(\frac{a^8}{100} - \frac{4a^7}{81} + \frac{6a^6}{64} - \frac{4a^5}{49} + \frac{a^4}{36}\right) \\ \quad + R_0^2 \left(\frac{1}{10} - \frac{4}{9} + \frac{6}{8} - \frac{4}{7} + \frac{1}{6}\right) \log(R_0), & r < R_0 \\ R_0^2 \left(\frac{1}{10} - \frac{4}{9} + \frac{6}{8} - \frac{4}{7} + \frac{1}{6}\right) \log(r), & r \geq R_0. \end{cases} \tag{99}$$

Our second test problem includes a tunable wavenumber component. This is useful to demonstrate how methods perform on charges with a high wavenumber which can only be resolved on fine grids. This second charge is defined as

$$\rho = \begin{cases} [\sin(wr)(\frac{r}{R_0} - \frac{r^2}{R_0^2})]^2, & r < R_0 \\ 0, & r \geq R_0. \end{cases} \tag{100}$$

We define w to be $\frac{2m\pi}{R_0}$ where m is a positive integer. The solution for this right hand side can be represented as

$$\phi = \begin{cases} \phi_{interior}, & r < R_0 \\ \phi_{exterior}, & r \geq R_0. \end{cases} \tag{101}$$

where the interior and exterior solutions are defined as

$$\begin{aligned}
\phi_{interior} &= \frac{\cos(2wr)-1}{w^2}\left(\frac{137}{64(wR_0)^4}-\frac{47a^2-52a+11}{32(wR_0)^2}\right) \\
&+ \frac{\cos(2wr)}{w^2}\frac{(a^4-2a^3+a^2)}{8} \\
&+ \frac{\sin(2wr)}{w^2}\left(\frac{77a-50}{32(wR_0)^3}-\frac{9a^3-14a^2+5a}{16wr}\right) \\
&+ \left(\frac{3}{16w^4R_0^2}-\frac{15}{16w^6R_0^4}\right)\left(\int\frac{\cos(2wr)}{r}\,dr-\log(r)\right) \\
&+ \left(\frac{3}{4w^5R_0^3}\right)\int\frac{\sin(2wr)}{r}\,dr \\
&+ \frac{r^4}{32R_0^2}-\frac{r^5}{25R_0^3}+\frac{r^6}{72R_0^4}-\frac{37R_0^2}{7200} \\
&+ \frac{R_0^2\log(R_0)}{120}\left(1+\frac{45}{w^4R_0^4}\right), \text{ and} \qquad (102) \\
\phi_{exterior} &= \frac{R_0^2\log(r)}{120}\left(1+\frac{45}{w^4R_0^4}\right) \qquad (103)
\end{aligned}$$

We are unable to write down a closed-form solution for this right hand side because it involves integration of terms such as $\frac{\sin(r)}{r}$ and $\frac{\cos(r)}{r}$. We generate integrals of these terms by numerical techniques, such as Taylor expansion near the singularity at the origin and Richardson-Romberg integration elsewhere [3]. These numerical approximations provide much higher order accuracy and are calculated with steps much smaller than the grid spacing used in our algorithm so that we can ensure that they are far more accurate than the $O(h^2)$ we expect from our algorithm. This extra accuracy is essential, since we compare the solutions calculated by our finite difference methods to this solution.

We have a third test problem for which no analytical solution is available. This charge distribution is a variation on the tunable wavenumber test problem above. The charge distribution varies in θ, as well as r.

We define this charge as

$$\rho = \begin{cases} [\sin(\frac{wr}{R^*})(\frac{r}{R^*}-\frac{r^2}{R^{*2}})]^2, & r < R^* \\ 0, & r \geq R^*. \end{cases} \qquad (104)$$

where w is defined as before and R^* is defined as

$$R^* = \frac{R_0\cos(6\theta)+3}{4}. \qquad (105)$$

The effect of the R^* term is to create lobes out from the center of the charge. Our particular choice of R^* creates a six-lobed charge field. Since no analytical solution is available for this charge distribution, Richardson error estimation must be used to study the convergence of the solution in this case.

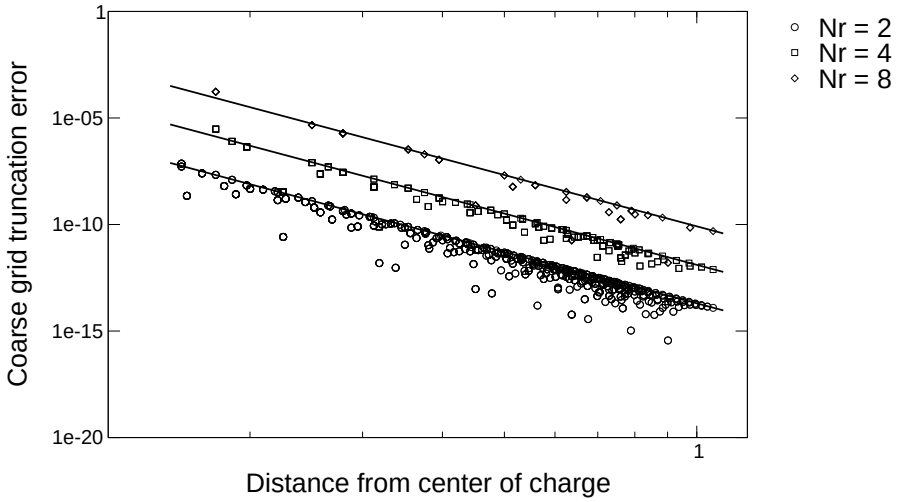

FIG. 3. Far-field coarse truncation error for a 65×65 fine grid.

4.2. Validation of Far-field Error Estimates

We show by numerical experiments that our analysis in Section 3.2 is correct. We are interested in confirming our estimates in (76) and (77), specifically that

$$\tilde{R}^H = L_9(Sample^H(\phi^h)) = O(H^6), \tag{106}$$

in the far field. Recall that this $\tilde{R}^H$ is measure of the error we make in our algorithm by truncating the far-field charge to zero when constructing the coarse charge field for the global coarse solve.

In order to verify this assertion, we use a test problem in which the charge has compact support, limited to a fraction of the solution domain. The charge that we use for these tests is given by (98), centered at $(\frac{1}{4}, \frac{1}{4})$, with a radius $R_0 = \frac{1}{16}$. The problem is solved on the domain $[0,1] \times [0,1]$ in $\Re^2$.

Figure 3 shows this far-field truncation error in the coarse grid representation of the charge for a fine grid domain of size 65×65. The curve fits in this truncation error plot are represented by

$$\tilde{R}^H = 3.64 \times 10^{-6} \frac{H^6}{|\vec{x}|^8}, \tag{107}$$

where H is the grid spacing used for the coarse grid Laplacian operator. This relationship implies that the coarse grid truncation error is $O(H^6)$. The variation of $\tilde{R}^H$ as $\frac{1}{|\vec{x}|^8}$ is also expected since the coefficients of $O(H^6)$ terms in the error estimate are eighth derivatives of the solution, and the solution is harmonic in the far field. Since the harmonic Green's function is proportional to $\log(|\vec{x}|)$, the eighth derivatives of a harmonic function are then $O(\frac{1}{|\vec{x}|^8})$.

To demonstrate that the error is a function only of coarse grid spacing and distance from the center of the charge, we have plotted the results for a 65×65 grid with $N_r = 2$, a 129×129 grid with $N_r = 4$, and a 257×257 grid with $N_r = 8$ in Figure 4. For each of these problems, the coarse grid spacing remains a constant

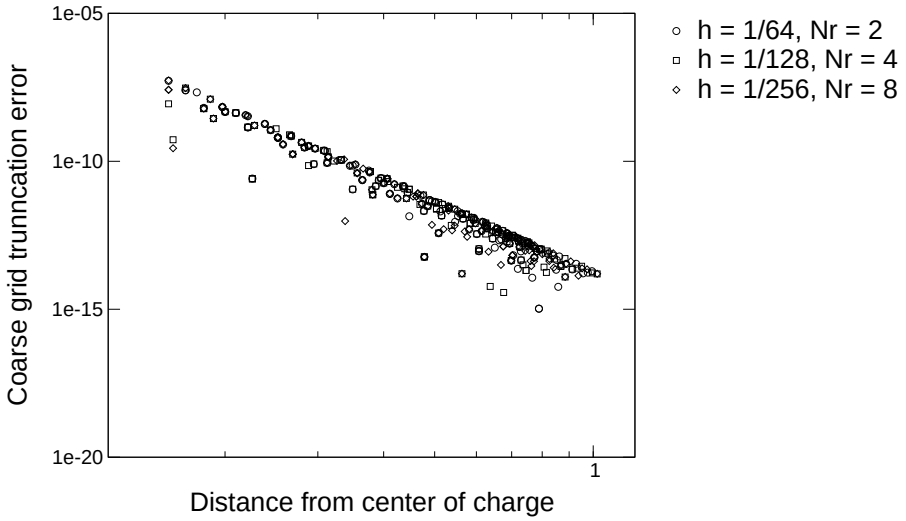

FIG. 4. Coarse grid truncation errors for various fine grid problems with a constant coarse grid spacing.

$H = \frac{1}{32}$, and all the data lie along a single line, independent of the fine grid spacing used to solve the Poisson problem.

4.3. Mesh Refinement Studies for the Full Algorithm

We now demonstrate by mesh refinement studies that our method produces results that are accurate to $O(h^2)$ as the solution grid is refined, and that the additional error induced by the domain decomposition is small, relative to the intrinsic solution error from a five-point discretization of Poisson's equation.

Accuracy for a High Wavenumber Charge

We use two test cases with high wavenumber components to gauge the performance of our method on non-smooth charge distributions. It is difficult to assess accuracy of a method on high wavenumber charges in terms of a standard convergence study: as the grid is refined, the problem becomes relatively more smooth. Therefore, for these tests, we restrict ourselves to three grid sizes: 129×129, 257×257, and 513×513.

The results for the radially symmetric test problem are shown in Table 1. For this set of test problems, the charge was defined by (100), with $R_0 = 0.42$ and $m = 15$. These parameters result in 30 periods over the radius of the charge, or approximately 1.8 grid points per period on the 129×129 grid, which is insufficient to resolve the solution. The 257×257 grid has adequate resolution, and the 513×513 grid is well resolved, with over 7 grid points per period. The convergence rates reported in Table 1 are based on comparing the 513×513 grid errors with the errors on the corresponding grids of size 129×129. These may be questionable since the 129×129 grid is under-resolved, but calculating the convergence based on the 513×513 and 257×257 grids gives $O(h^2)$ convergence uniformly as well.

An error plot for this problem calculated on a 257×257 grid with $N_p = 2$ and $N_r = 8$ is shown beside an error plot for the same problem solved on a single 257×257 grid in Figure 5. The plots are very similar, and, equally important,

TABLE 1
Error norms and convergence rates for a problem with a high wavenumber charge distribution, solved with various grid sizes, numbers of patches, and refinement ratios.

			Error Norms		Convergence Rates	
Grid Size	N_p	N_r	$\|\text{error}\|_{\max}$	$\|\text{error}\|_2$	$\|\text{error}\|_{\max}$	$\|\text{error}\|_2$
129×129	1	2	4.42e-7	9.21e-8		
129×129	1	4	4.41e-7	9.18e-8		
129×129	2	4	4.91e-7	1.03e-7		
129×129	2	8	4.83e-7	1.00e-7		
257×257	2	4	8.61e-8	2.18e-8		
257×257	2	8	8.51e-8	2.13e-8		
257×257	4	8	8.25e-8	2.02e-8		
257×257	4	16	7.23e-8	1.77e-8		
513×513	2	4	2.02e-8	5.32e-9	2.22	2.06
513×513	2	8	2.01e-8	5.26e-9	2.23	2.06
513×513	4	8	1.96e-8	5.05e-9	2.32	2.18
513×513	4	16	1.67e-8	4.12e-9	2.42	2.30

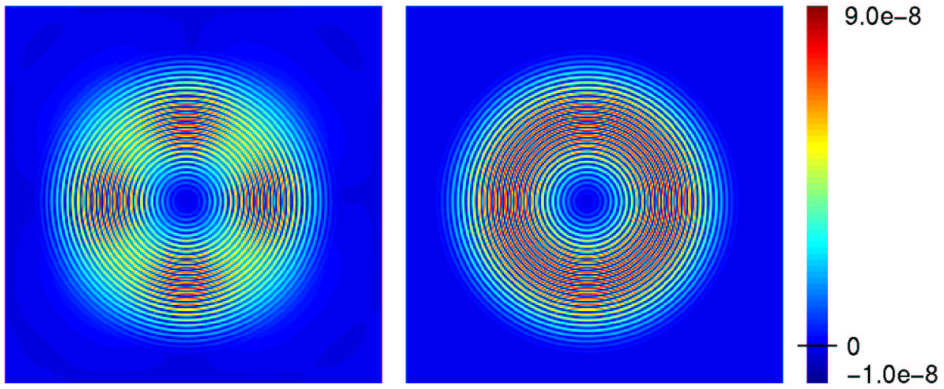

FIG. 5. On the left, error for a high wavenumber, radially symmetric charge covering most of the solution domain, a 257×257 grid, with $N_p = 2$, and $N_r = 8$; on the right, the error resulting when the same problem is solved on a single 257×257 grid.

TABLE 2
Error norms for a single grid solution of the high wavenumber problem.

Grid Size	$\|\text{error}\|_{\max}$	$\|\text{error}\|_2$
129×129	4.92e-7	1.19e-7
257×257	8.62e-8	2.42e-8
513×513	2.02e-8	5.83e-9

the domain decomposition algorithm does not add any strong discontinuities to the error. The error norms for the single grid solutions are shown in Table 2.

To be fair, however, we should also compare our results with results from an infinite domain solver which uses Mehrstellen preconditioning: we need to demonstrate that a much simpler, single grid solution on a much coarser grid could not produce equally accurate results. In Table 3 we juxtapose the previous results from our algorithm with results from an infinite domain solution using Mehrstellen preconditioning (29). The MLC algorithm is used with the fine and coarse grid sizes as noted in the table; the corresponding single grid infinite domain Poisson solution with Mehrstellen preconditioning is performed on grid with the coarse spacing. Note that in some cases, there are multiple values of N_p for the grid sizes and refinement ratios indicated in the table: in every such case we have chosen the result which reflects the largest errors for our algorithm. We see that for any N_r we choose, our method is more accurate than any single grid solution with Mehrstellen preconditioning on the corresponding coarse grid domain. Until the problem is sufficiently resolved on the grid, the benefits of a higher order method with Mehrstellen preconditioning are not realized.

Accuracy for an Extremely Large Problem

TABLE 3
Comparison of max error norms for a single coarse grid solution of the high wavenumber problem, utilizing Mehrstellen preconditioning, to error norms for MLC.

			MLC	Mehrstellen	
Fine Size	N_r	Coarse Size	$\|\text{error}\|_{\max}$	$\|\text{error}\|_{\max}$	Ratio: $\frac{\text{MLC}}{\text{Mehrstellen}}$
129×129	2	65×65	4.42e-7	1.13e-5	3.9e-2
129×129	4	33×33	4.91e-7	5.97e-4	8.2e-4
257×257	4	65×65	8.61e-8	1.13e-5	7.6e-3
257×257	8	33×33	8.51e-8	5.97e-4	1.4e-4
513×513	4	129×129	2.02e-8	1.75e-7	1.2e-1
513×513	8	65×65	2.01e-8	1.13e-5	1.8e-3
513×513	16	33×33	1.67e-8	5.97e-4	2.8e-5

The ability to solve large problems is one of the main reasons for developing parallel algorithms. Unfortunately, as problem sizes get larger and charge distributions get more complicated, it becomes more difficult to study convergence of the solution. Here we solve a rather complicated problem on grids of size 513×513, 1025×1025, and 2049×2049, testing the accuracy of our solutions on the 513×513 and 1025×1025 grids by using Richardson error estimation.

The charge distribution for this large problem is the superposition of three separate high wavenumber charge fields and a smooth, radially symmetric field. Two of the high wavenumber charge fields are six-lobe charge fields as defined by (104); the third is a radially symmetric charge field defined by (100). The smooth, radially symmetric charge field is defined by (98). The problem domain is defined on $[0,1] \times [0,1]$. One six-lobe charge with $m = 6$ and $R_0 = 0.25$ is placed at $(0.35, 0.3)$. A second six-lobe charge with $m = 4$ and $R_0 = 0.18$ is placed at $(0.75, 0.8)$. The high wavenumber, radially symmetric charge has $m = 18$ and $R_0 = 0.38$ and is placed at $(0.6, 0.4)$. The smooth, radially symmetric charge is placed at $(0.4, 0.5)$ and also has $R_0 = 0.38$. Error norms for this large problem are listed in Table 4: again the method exhibits $O(h^2)$ accuracy.

A plot of the error on the 1025×1025 grid appears in Figure 6. It is possible to see faint traces along the domain boundaries in this plot. These artifacts are very small in magnitude compared to the intrinsic error due to the five-point discretization of the Poisson equation.

4.4. Parallel Timing

We would like to confirm experimentally that our algorithm scales well as the number of processors is increased. There are many different measures of parallel speed-up, depending on the type of problem being solved. For some problems

TABLE 4
Error norms for a very large, high wavenumber problem.

Grid Size	N_p	N_r	$\|\text{error}\|_{\max}$
513×513	2	8	2.97e-8
1025×1025	4	16	6.47e-9

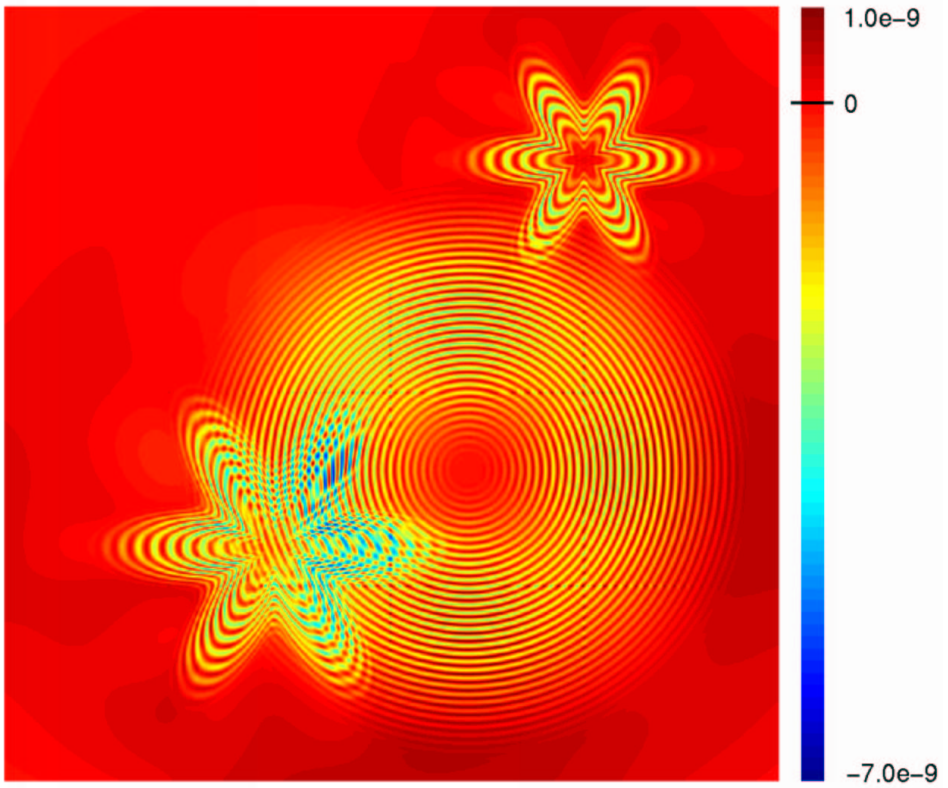

FIG. 6. Error for a high wavenumber charge, solved on a 1025×1025 grid with 16 patches and $N_r = 16$.

TABLE 5
Scaled Speed-up Test: Series 1.

Total Size	Patches	Refinement Ratio	Local Fine Size	Global Coarse Size
129×129	1×1	4	129×129	33×33
257×257	2×2	8	129×129	33×33
513×513	4×4	16	129×129	33×33

multiple processors are used in order to take advantage of parallelism inherent in the problem. For such problems, one hopes to compute solutions to the same problem in an amount of time inversely proportional to the number of processors available. For another class of problems, however, multiple processors are needed because the data is too large to fit in the memory associated with a single processor. For these types of problems, scaled speed-up is a more appropriate measure of parallel performance. In scaled speed-up tests, the size of the problem is scaled in proportion to the number of processors, and one hopes that the time to compute solutions remains constant even as the problem size increases.

Our algorithm is designed for scaled speed-up. As discussed in Section 3.5, our method introduces some degree of computational overhead even when solving Poisson's equation with infinite domain boundary conditions. If our method were used instead to solve problems with simpler Dirichlet boundary conditions, the overhead would be an even greater factor. As a result, single-grid methods are preferable when problems are small enough to fit on a single processor. As problem sizes grow, however, and parallelism is necessary, the scaled speed-up provided by our algorithm becomes more attractive.

We constructed series of test problems for which the amount of computation required per processor remains approximately constant. For this performance study, we expect the wall-clock time required to be nearly constant as the problem sizes and number of processors increase.

The amount of computation required per processor depends on both the local fine grid patch sizes and the global coarse grid size. Therefore, as we increase the overall problem size, N, we need to increase both the refinement ratio, N_r, and the number of patches in each direction, N_p. Three series of test problems were run on 1 to 16 processors. The parameters for these series are shown in Tables 5, 6, and 7.

The results of the first groups of scaled speed-up tests, shown in Figure 7 are very near what we would expect. The three series lie nicely along horizontal lines. In fact, the deviations from perfectly horizontal lines can be explained almost entirely by variations in the total number of multigrid iterations required.

We are not only interested in the total time required to compute solutions, however. We are also interested in the fraction of time spent communicating data among the processors. The communication overhead for our algorithm, shown in

TABLE 6
Scaled Speed-up Test: Series 2.

Total Size	Patches	Refinement Ratio	Local Fine Size	Global Coarse Size
257×257	1×1	4	257×257	65×65
513×513	2×2	8	257×257	65×65
1025×1025	4×4	16	257×257	65×65

TABLE 7
Scaled Speed-up Test: Series 3.

Total Size	Patches	Refinement Ratio	Local Fine Size	Global Coarse Size
513×513	1×1	4	513×513	129×129
1025×1025	2×2	8	513×513	129×129
2049×2049	4×4	16	513×513	129×129

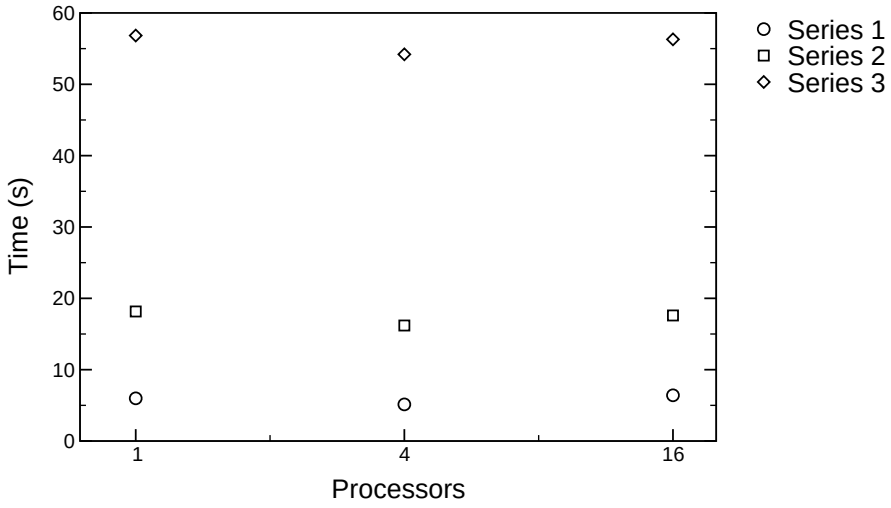

FIG. 7. Timing results for a scaled speed-up test.

TABLE 8
Communication costs for a series with $O(h^2)$ convergence.

Problem Size	Patches	N_r	Communication Time (s)	Percent of Total
257×257	2×2	8	0.081	1.5
513×513	2×2	8	0.17	1.1
513×513	4×4	16	0.16	2.5
1025×1025	2×2	8	0.31	0.57
1025×1025	4×4	16	0.30	1.7
2049×2049	4×4	16	0.54	0.96

Table 8, is very small, and never more than 2.5% of the total time for the method.

5. CONCLUSIONS

We developed a parallel finite difference algorithm for the computation of the solution to the Poisson equation with infinite domain boundary conditions.

Our algorithm produces results accurate to second-order for a wide range of problems. The errors in the computed solution are almost entirely a function of fine grid spacing. The division of the problem into subdomains produces additional errors at the boundaries which are small in magnitude compared to the errors on the interiors of the subdomains. In fact, the overall errors for solutions calculated by our method are comparable to those produced by a standard second-order single grid solver.

The accuracy of the solution is not significantly degraded even for very coarse global grids. We have found that it is possible to use coarse grid spacings large enough that the coarse grid solve represents only about a 2-7% increase in the total number of floating point operations.

The communication required is small. Time spent in communication represents no more than 2.5% of the total time required to compute a solution for the Titanium implementation of the algorithm running on the IBM SP2 at the San Diego Supercomputing Center. Since both our parallel computational overhead costs and our communication requirements are small, we expect our algorithm to exhibit very good parallel scaling. We find that our program does obtain linear speed-up on the SP2.

An important extension of this research will be the development of the three-dimensional analogue of this algorithm. Unlike many other approaches, which become much more complex or much less efficient in the transition from two to three dimensions, our algorithm should remain fairly simple and largely unchanged. The most significant change required has to do with the potential calculation for the infinite domain solve: a straightforward extension of our method to three dimensions

would result in a method which required $O(N^4)$ work to compute the boundary charge on a grid of N^3 points. This method would be unacceptable since all other phases of the computation would be expected to require $O(N^3 \log N^3)$ work at most. Projecting the potential onto a coarser outer grid, of spacing $H = O(\sqrt{h})$, would reduce the work required to $O(N^3)$, again, though, and $O(H^8)$ interpolation on the outer grid to set the boundary condition would be tedious, but not computationally expensive.

Another important addition to our algorithm would be the introduction of alternate boundary conditions. Many real-world problems require Dirichlet, Neumann, or some sort of mixed boundary conditions that our algorithm is currently not designed to handle. One approach to supplying alternate boundary conditions to problems such as these uses images charges. An example of this approach is described by Baden and Puckett [6].

It would also be interesting to look at making the algorithm adaptive: we have seen that the accuracy of our method depends very weakly on the coarse grid spacing, and it should be possible to use coarser local grids on patches with low wavenumber charge distributions. Adaptivity such as this would require us to look into load-balancing issues much more carefully, however.

ACKNOWLEDGMENTS

We thank Scott Baden for his help in implementing early versions of this work in KeLP. We thank the Titanium group at the University of California, Berkeley for their help in analyzing and improving the Titanium version of our code, and especially Katherine Yelick for her advice in designing parallel code.

Support for this work was provided by the following grants and fellowships: "National Science Foundation Graduate Fellowship Program," National Science Foundation; "The Berkeley Fellowship," University of California at Berkeley; "Adaptive Numerical Methods for Partial Differential Equations," US Department of Energy Mathematical, Computing, and Information Sciences Division, Grant DE-FG03-94ER25205; "Computational Fluid Dynamics and Combustion Dynamics," US Department of Energy High-Performance Computing and Communications Grand Challenge Program, Grant DE-FG03-92ER25140.

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under contract F30602-95-C-0136. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

REFERENCES

1. A. S. Almgren, T. Buttke, and P. Colella. A fast adaptive vortex method in three dimensions. *Journal of Computational Physics*, 113(2):177–200, August 1994.
2. C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *Journal of Computational Physics*, 62:111–123, 1986.
3. B. M. Ayyub and R. H. McCuen. *Numerical Methods for Engineers.* Prentice Hall, Upper Saddle River, New Jersey, 1996.
4. S. B. Baden. *Run-Time Partitioning of Scientific Continuum Calculations Running on Multiprocessors.* PhD thesis, University of California, Berkeley, 1987.
5. S. B. Baden. Software infrastructure for non-uniform scientific computations on parallel processors. *Appled Computing Review*, 4(1):7–10, Spring 1996.
6. S. B. Baden and E. G. Puckett. A fast vortex method for computing 2d viscous flow. *Journal of Computational Physics*, 91(2):278–297, December 1990.
7. G. T. Balls. *A Finite Difference Domain Decomposition Method Using Local Corrections for the Solution of Poisson's Equation.* PhD thesis, University of California, Berkeley, 1999.
8. L. Collatz. *The numerical treatment of differential equations.* Springer-Verlag, 3rd edition, 1966.

9. S. J. Fink, S. B. Baden, and S. R. Kohn. Efficient run-time support for irregular block-structured applications. *Journal of Parallel and Distributed Computing*, 50(1-2):61–82, 1998.

10. L. Greengard and J. Y. Lee. A direct adaptive solver of arbitrary order accuracy. *Journal of Computational Physics*, 125(2):415–424, May 1996.

11. L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *The Journal of Computational Physics*, 73:325–348, 1987.

12. L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269, 1997.

13. R. A. James. The solution of Poisson's equation for isolated source distributions. *Journal of Computational Physics*, 25(2):71–93, October 1977.

14. K. Lackner. Computation of ideal MHD equilibria. *Computer Physics Communications*, 12(1):33–44, 1976.

15. B. F. Smith and O. B. Widlund. A domain decomposition algorithm using a heirarchical basis. *SIAM Journal on Scientific and Statistical Computing*, 11(6):1212–1220, November 1990.

16. J. Strain. Fast potential theory. II. Layer potentials and discrete sumes. *Journal of Computational Physics*, 99(2):251–270, April 1991.

17. K. Yelick, L. Semenzato, G. Pike, C. Miyamoto, B. Liblit, A. Krishnamurthy, P. Hilfinger, S. Graham, D. Gay, P. Colella, and A. Aiken. Titanium: A high-performance Java dialect. *Concurrency—Practice and Experience, Java Special Issue*, 1998.